Implementing Configuration Management for Software Testing Projects©

Steve Boycan
Securities Industry Automation Corporation

Dr. Yuri Chernak
Valley Forge Consulting, Inc.

This case study presents the Application Scripting Group's experience in implementing the configuration management (CM) process for critical software testing projects. The article describes the company's test process management objectives and how implementing the CM process helped testers better achieve them. The authors define the types and purposes of the test process milestones and the corresponding types of test model baselines, and describe the CM process implementation with the Rational ClearCase tool.

This case study discusses software testing of use-case-driven projects by the Application Scripting Group (ASG) of the Securities Industry Automation Corporation (SIAC). This SIAC group is responsible for testing highly critical systems used for equity trading at the New York Stock Exchange (NYSE). The high criticality of SIAC systems requires that the software testing process be well planned and well managed. In addition, an important management objective is the continuous improvement of test process performance that focuses on test effort estimation, completeness of test designs before test execution, traceability of test designs to use cases, testing effectiveness in finding defects, and test artifact maintainability and reusability from one project cycle to another. This means that when a given project has ended, a project team has to analyze the actual process performance, perform causal analysis of process deficiencies, and identify improvements for the next project cycle.

To analyze test process performance, testers typically review and analyze the test process artifacts produced and used during a project cycle. However, these testing artifacts, along with their related use cases, evolve during a project cycle and can frequently have multiple versions by project end. Hence, analysis of the process performance from different perspectives requires that testers know exactly which versions of artifacts they used for different tasks. For example, to analyze why the test effort estimates were not sufficiently accurate, testers need the initial versions of use cases, test analysis, and test design specifications they used as a basis for the effort estimation. In contrast, a causal analysis of software defects missed in testing requires testers to have the latest versions of use cases, test analysis, and test design specifications used in test execution.

As this case study illustrates, implementing a configuration management (CM) process provides an effective solution to this issue. It allows testers to capture the versions of their artifacts and the related versions of use cases to produce configuration baselines that can effectively support the analysis of test process performance after the project cycle has ended. In addition, during a project cycle this kind of CM process provides management with much better visibility into and control over the test process. This article describes how the ASG defined their CM process and implemented it for use-case-driven projects with IBM's Rational ClearCase tool. However, the discussed CM process is generic and can be implemented with any CM tool available on the market.

Defining the Test Process Workflows

ASG's implemented CM process was intended to support the test process and make it more efficient and better controlled. Hence, before discussing the CM process, we need to explain how we defined the test process. The Rational Unified Process (RUP) methodology [1] defines the test process as one of its nine disciplines. When testers deal with complex use-case models, they can benefit from further decomposing the RUP's test discipline into six separate workflows shown in Figure 1 and defined in the following paragraphs. As we found on our projects, such test process decomposition can help software testers better cope with functional complexity of software systems, and it can help management better control a testing project.

Workflow 1: Test Analysis and Planning

- **Purpose:** The objectives of this workflow are to define the test strategy and testing objectives for each level of testing, to analyze the use-case model to determine the testing scope and priorities, to provide test effort estimates, to allocate project resources, to analyze quality risks within the context of use-case scenarios, and to develop test ideas about what must be tested for each use case.
- **Key Resulting Artifacts:** A test project plan, system test plan, test automation plan, test guidelines, and test analysis specifications.

Workflow 2: Testware Design and Maintenance

- **Purpose:** The objectives of this workflow are to refine test ideas about what must be tested for each use case and provide details about how to execute these tests. Also, this workflow includes maintenance of the existing test designs.
- **Key Resulting Artifacts:** Test analysis, test design, and test case specifications (and/or test procedure specifications).

Workflow 3: Test Preparation

- **Purpose:** The objectives of this workflow are to set up a test environment and a defect tracking system, generate required test data, and develop test supporting utilities (if required).
- **Key Resulting Artifacts:** Requirements for the test environment, guidelines for test data generation and management, the actual test environment and test data ready for use, and test supporting utilities (if required).

Workflow 4: Test Execution and Reporting

- **Purpose:** The objectives of this

© Copyright 2005 by the Securities Industry Automation Corporation (SIAC). All rights reserved. Except as permitted under the United States Copyright Act of 1976, no part of this document may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the Securities Industry Automation Corporation.

workflow are to execute tests and evaluate the quality of the software product, find and report software defects, and report the product's testing progress and status.
- **Key Resulting Artifacts:** Test analysis and test design specifications (enhanced, for example, with *exploratory* test ideas), software defect reports, test execution logs, and test execution status reports.

Workflow 5: Test Process Evaluation
- **Purpose:** The objectives of this workflow are to evaluate the test process completeness, effectiveness, and efficiency; perform a causal analysis of *test escapes*[1]; and provide recommendations for the test process improvement.
- **Key Resulting Artifacts:** A test summary report, collected test process and product metrics, and a process improvement report (that can include findings of the test escape causal analysis and post-implementation review).

Workflow 6: Regression Test Automation
- **Purpose:** The objectives of this workflow are to develop the test automation architecture and automated regression scripts.
- **Key Resulting Artifacts:** Test automation documentation and test automation software, i.e., automated regression scripts.

Workflows in the Project Cycle
These six test process workflows can, as do all other RUP workflows, overlap in time and iteratively evolve throughout the project cycle (see Figure 1). For example, ASG testers are expected to continue exploring the software product during test execution to develop additional test ideas. Hence, the Test Analysis and Planning and Testware Design and Maintenance workflows largely overlap with the Test Execution and Reporting workflow for this reason.

We have already mentioned three types of test documentation used on ASG projects: test analysis, test design, and test case specifications. Now, we need to explain their purposes. The test analysis specification is developed for each use case. It provides analysis and decomposition (slicing) of a given use case and identifies its quality concerns to be addressed in testing. The test design document is also developed for each use case. It captures a high-level testing logic and rationale for test case selection for each of the quality concerns identified in the corresponding test analysis specification. On ASG projects, testers primarily use the test design specifications for manual test execution. Lastly, the test case specifications focus on the how to execute testing details and are primarily intended for the test automation personnel that use them as functional specifications for developing automated regression scripts.

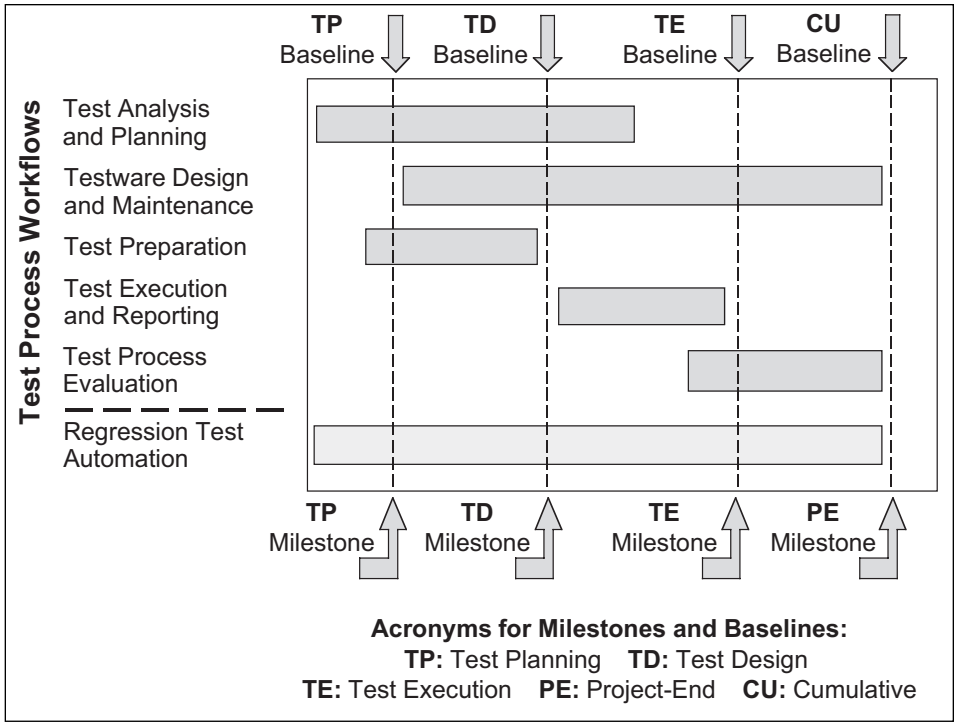

Figure 1: *Project Milestones and Related Baselines*

As Figure 1 shows, the Regression Test Automation workflow spans the entire testing project. In our case, it is managed as a separate project with its own project plan and personnel possessing specialized programming skills. The artifacts of this workflow, i.e., test automation documentation and regression scripts, are also considered a part of the entire test model, which is a collection of all testing artifacts [1].

Defining the Configuration Management Process
According to our CM process, test model baselines are intended to support the test process milestones. For this reason, there is a one-to-one relationship between the baselines and milestones as Figure 1 shows. This section discusses how we defined the types of test project milestones and their corresponding baselines.

Test Project Milestones
The term *milestone* sometimes has different meanings on different projects. According to Webster's dictionary, milestone is defined as "a significant point in development." In the case of ASG, a project milestone means a significant point in the test project life cycle where a test team completes a critical task and makes an important project decision. Thus, ASG defined its project milestones as follows:
- **Test Planning (TP) Milestone.** The test team decides that the test project is feasible and confirms the test project plan and schedule. Testers make this decision based on the completed test analysis and test effort estimation.
- **Test Design (TD) Milestone.** Testers decide that they have completed their test designs sufficiently enough to start test execution.
- **Test Execution (TE) Milestone.** The test team decides to stop testing. Testers make this decision based on evidence that they have met the defined test exit criteria.
- **Project-End (PE) Milestone.** The test team finishes the project cycle and decides which of the test model artifacts should be maintained and reused in future project cycles.

As we already mentioned, the decisions made at project milestones will be evaluated later as part of the project performance improvement task. Hence, testers need to capture the versions of the test model artifacts that supported each of the project milestones during a project cycle. An identified and fixed configuration of these versions is called

Test Model Artifacts	TP Baseline	TD Baseline	TE Baseline	CU Baseline
Component 1- Manual Testing Artifacts				
Test Plan	X	X	X	
Test Analysis Specifications	X	X	X	X
Test Design Specifications		X	X	X
Test Execution Logs			X	
Test Summary Report			X	
Baseline Status Reports	X	X	X	X
Component 2 - Automated Testing Artifacts				
Automation Project Plan	X			X
Test Case Specifications				X
Script Design Specifications				X
Automation Infrastructure Designs				X
Automated Regression Scripts				X
Automation Infrastructure Application Program Interfaces				X

Table 1: *Mapping the Test Model Artifacts to Their Baseline Types*

baseline, and the next section discusses how ASG defined the baseline types.

Test Model Baselines

ASG defined the following types of test model baselines:

- **TP Baseline.** This baseline supports the TP milestone. It captures the version of a test plan that identifies use cases in the scope of the testing project. Also, it captures the versions of test analysis specifications that testers use as a basis for the test effort and schedule estimation.
- **TD Baseline.** This baseline supports the TD milestone. It provides evidence that testers sufficiently completed test designs and can start test execution. It captures the versions of test analysis and test design specifications that have been completed based on use cases and other available functional specifications, and that have been updated based on the peer-review findings. In addition, this baseline includes the latest version of a test plan document.
- **TE Baseline.** This baseline supports the TE milestone. It captures information about how the system was tested that testers deem necessary to support their conclusion about the software product's quality. In particular, it includes the latest versions of test analysis and test design specifications[2] that have evolved during the manual test execution. In addition, it includes the test execution logs, both manual and automated, and a test summary report.
- **Cumulative (CU) Baseline.** This baseline supports the PE milestone. It captures test assets intended for maintenance and reuse in subsequent project cycles. This baseline has two components: manual testing artifacts and automated testing artifacts. Unlike the other baseline types that capture artifacts created and used only in a given project cycle, the CU baseline captures the cumulative set of artifacts created both in the current project cycle and in previous cycles[3].

Finally, different baseline types can be composed of different test model artifacts. Table 1 shows the mapping between the test artifacts and their corresponding baseline types established for the ASG projects.

Implementing the CM Process

This section discusses the CM process reference model, which was selected for the ASG projects, and explains in detail the CM process implementation.

A CM Process Reference Model

ASG implemented the CM process by following the practices of the Software Configuration Management Key Process Area defined in the Software Engineering Institute's Capability Maturity Model® (CMM®) framework [2]. To better manage the CM process implementation, we further grouped the CMM practices by four categories that compose the conventional CM discipline [3]:

- Configuration Identification.
- Configuration Control.
- Configuration Status Accounting.
- Configuration Auditing.

The implementation of the CM process began by producing a general document for the department that defined a CM policy, glossary of CM terms, a standard CM process, and guidelines for its implementation. Then, based on this document, each project team developed its own CM plan document that followed the Institute of Electrical and Electronics Engineers (IEEE) Std.828-1998 "IEEE Standard for Software Configuration Management Plans." These CM plans were project-specific and defined configuration items and their naming conventions, the CM repository structure, and the team member roles and responsibilities. In particular, each project assigned the CM manager role to one of the team members. Finally, in implementing the CM process, the project teams used the CM plans as a basis for performing their CM activities.

Configuration Identification

When performing the configuration identification activities, a project team decides which test model artifacts must be under configuration control, what should be in a team-shared repository, and how the repository should be structured. On use-case-driven projects, use-case models can form complex structures via the include/extend relationships [4]. In this case, different testers can be assigned to produce test designs for related use cases. Hence, it is important that team members have quick access in the course of a project to the latest versions of each other's test documentation. Establishing a common CM repository of the test model artifacts provides an effective solution to this issue.

ASG projects created their CM repositories using IBM's Rational ClearCase tool. Each repository contained a set of directories, each storing a particular type of configuration item. For example, all test design specifications, created by the project team, were stored in the same directory. Table 1 shows the artifacts of the test model that we included in the configuration control. As you can see, they are logically divided into two components: manual testing artifacts and automated testing artifacts that, in turn, include the automation design documentation and software, i.e., automated regression scripts.

ASG then defined a naming convention for all artifacts under configuration control. Here is a file name example of a test design specification that illustrates our naming convention: TDS_DB_9.1 .doc. In this example, the file name is composed of the following parts:

- **Artifact type:** TDS, meaning the arti-

® Capability Maturity Model and CMM are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

fact type *test design specification*.
- **Project code:** DB, meaning the project name *Display Book*.
- **Use Case ID number:** 9.1.

By our convention, use cases and test design specifications have a one-to-one relationship. Hence, by including the use case number in the test design file name, ASG established an explicit traceability from use cases to their corresponding test designs.

Configuration Control

Performing the configuration control activities involves controlling change requests for configuration items, reporting and tracking problems, controlling versions of configuration items, capturing the change history each time a new version is created, labeling artifacts of a test model, and creating its baselines associated with various project milestones. A project manager is the primary source of change requests. He/she informs the testers when a new project cycle begins, when they should start working on test designs, which new automated scripts should be created, etc. In addition, any project team member who finds a problem with automated scripts reports it via a defect tracking system.

The version control of test project artifacts was handled by the CM tool that allows the creation of new versions of a given artifact and the capture of its change history notes, the date/time when the new version was created, the owner of this version, and so on. For some of the artifact types, ASG also created custom attributes in ClearCase[4]. For example, for the test design specifications we created attributes to capture the peer-review date and the total number of test cases for a given test design. This information provided management with better visibility into the state of evolving test designs. Also, the number of test cases was captured as part of the historical project data for evaluating and improving the test execution effort estimation.

Creating a test model baseline is a three-step process. First, the CM manager creates labels that correspond to the project milestones discussed earlier. Second, the artifact versions intended for inclusion in a particular baseline are labeled to correspond to the requested baseline. For the TP and TD baselines, the artifact owner is responsible for labeling versions of his/her own work products, as only the owner knows when an artifact is ready for inclusion in either the TP or TD baseline. In contrast, applying labels at the end of test execution (TE baseline) or at the end of the entire project cycle (CU baseline) can be done for all required artifacts at the same time. Hence, in the case of TE and CU baselines, the CM manager can be responsible for labeling the test model artifacts by simply running a script. Finally, at the third step the CM manager locks the label and completes the baseline creation by producing and publishing a baseline status report. Following these steps allows creating the required baselines and capturing the versions of evolving test model artifacts at different points in the project life cycle.

Configuration Status Accounting

The main objective of the ASG status accounting activities was to verify and report to a team and its management that a given baseline is compliant with its completion criteria. The CM manager was responsible for this task. First, before creating each baseline, this task required verifying that the set of test model artifacts was compliant with the baseline completion criteria. Second, after a given baseline has been created, the task required producing and publishing a baseline status report. As different test model baselines have different purposes and different contents shown in Table 1, ASG defined the completion criteria for each of them as follows:

"By following the defined CM process, our testing teams established their team-shared project repositories, implemented effective version control of their artifacts, and produced test model baselines to support different project milestones."

TP Baseline

The TP baseline (*captures information used for the test effort estimation*) requires the following:
- A test plan has been reviewed and its current version has been labeled.
- All required test analysis specifications exist in the CM repository.
- The latest versions of test analysis specifications, used for the test effort estimation, have been labeled and refer to the corresponding use-case document versions.
- A baseline status report has been created and labeled.

TD Baseline

The TD baseline (*captures test designs to be used for test execution*) requires the following:
- The latest version of the test plan document has been labeled.
- All required test analysis and test design specifications have been completed and peer-reviewed.
- The latest versions of test analysis and test design specifications, ready for test execution, have been labeled.
- All labeled versions of test analysis and test design specifications refer to the corresponding use-case versions.
- All labeled test design specifications capture the peer-review date and the total number of test cases.
- A baseline status report has been created and labeled.

TE Baseline

The TE baseline (*captures test artifacts actually used for test execution*) requires the following:
- The latest version of the test plan document has been labeled.
- The latest versions of test analysis and test design specifications, actually used for test execution, have been labeled.
- All labeled versions of test analysis and test design specifications refer to the corresponding use-case versions.
- All labeled test design specifications capture the total number of test cases (that may have increased during test execution).
- The test execution logs (manual and automated) exist in the CM repository and their latest versions have been labeled.
- The test summary report exists in the CM repository and its latest version has been labeled.
- A baseline status report has been created and labeled.

CU Baseline

The CU baseline (*captures test artifacts intended for reuse and maintenance*) requires the following:
- The latest versions of all selected test

analysis specifications have been labeled.
- The latest versions of all selected test design specifications have been labeled.
- The latest version of the test automation plan has been labeled.
- The latest versions of all selected test case specifications have been labeled.
- The latest versions of test automation documentation have been labeled.
- The latest versions of automated scripts that have been accepted for production have been labeled.
- Each accepted automated script captures the date it was accepted for production and the name of a team member who tested and accepted the script.
- A baseline status report has been created and labeled.

Configuration Auditing

Baseline auditing was an important part of our CM process. Especially in the beginning of the process implementation a risk existed that some team members, because of their lack of experience, might not follow the process exactly as defined. To ensure that the CM process is properly followed, the ASG department established a quality assurance (QA) function. The QA personnel oversaw all of the process improvement tasks in the department, including the CM process implementation and project baseline audits.

Each testing project plan included baseline audit tasks and assigned a QA auditor as a project resource responsible for the task. By having access to each project team's configuration repository, the auditor could verify that each created baseline was compliant with its defined completion criteria. In addition, the auditor was monitoring each project team's CM activities during a project cycle. Thus, any deviations from the process then could be identified and corrected earlier in the project. The audit findings were reported to project teams and discussed at their status meetings. Also, the audit summary report was periodically submitted to and reviewed by the department head.

CM Process Implementation Challenges

As is the case with any software process improvement implementation, while implementing our new CM process we experienced a few challenges. These challenges can be described from two perspectives: (1) a general process improvement perspective, and (2) a CM process-specific perspective.

From the general process improvement perspective, the success of any new process implementation depends not only on a good definition of the process activities and procedures, but also on a number of other (process supporting) critical factors. Among them, the most important are establishing process ownership and leadership, allocating necessary resources, personnel training, management reviews, and process auditing. Because of this dependency, we found that the practices defined in the CMM as the following common features [2] – c*ommitment to perform, ability to perform, measurement and analysis, and verifying implementation* – are all equally important for successful process implementation as the practices in the main CMM category – *activities to perform*.

From the CM process-specific perspective, before a project team starts implementing a new CM process, it should decide how to handle legacy test model artifacts when moving them into the new CM repository and creating an initial project baseline. Likely, these artifacts will not be in compliance with the defined CM process requirements. For example, old test designs might not have their peer review dates and/or references to their corresponding use-case versions. Likewise, old automated scripts might not have their acceptance dates, etc. Hence, a newly created baseline might not satisfy its completion criteria when it includes the legacy configuration items.

One way to resolve this issue is to establish a convention defining how the missing metadata can be substituted while moving the legacy artifacts to the new CM repository. This way a status report – used to determine the baseline's completion – will not have blank values for the metadata pertinent to the legacy configuration items.

Conclusion

This case study discussed our experience with implementing the CM process for highly critical software testing projects. An important management objective on our projects is to establish a framework for (a) effective control of testing projects, and (b) continuous analysis and improvement of test process performance. As we illustrated in this article, implementing the CM process allows software testers to better achieve this management objective. By following the defined CM process, our testing teams established their team-shared project repositories, implemented effective version control of their artifacts, and produced test model baselines to support different project milestones. These baselines captured configurations of versions of the testing artifacts and their related use cases that could later support the testers' analysis and improvement of test process performance. Finally, the discussed CM process is generic and can be implemented with any available CM tool.◆

References

1. Kroll, P., and P. Kruchten. The Rational Unified Process Made Easy: A Practitioner's Guide to the Rational Unified Process. Addison-Wesley Professional, 2003.
2. Paulk, M., et al. The Capability Maturity Model: Guidelines for Improving the Software Process. Addison-Wesley Professional, 1995.
3. Ben-Menachen, M. Software Configuration Management Guidebook. McGraw-Hill Book Company, 1994.
4. Bittner, K., and I. Spence. Use Case Modeling. Addison-Wesley, 2003.

Notes

1. From a system test perspective, *test escape* is a software defect missed in system testing and found in a later stage, for example, during independent quality assurance testing or in production.
2. Capturing the versions of test designs at this point is critical to supporting at a later time the causal analysis of test escapes, especially those found in production.
3. Regression Test Automation workflow can have its own intermediate baselines; however, the versions of these workflow deliverables must be synchronized with other test model artifacts at the end of a project cycle (in the CU baseline)
4. This ClearCase feature – adding custom attributes to configuration items – may not be available in some commercial CM tools.

Acknowledgements

The authors are grateful to the CROSSTALK reviewers and Robin Goldsmith at GoPro Management for their feedback and comments that helped us improve this article. Our special thanks to the ASG testers who have implemented and followed the CM process discussed in this article, and helped us refine the process and make it effective.

About the Authors

Steve Boycan is managing director for Process Improvement and Software Engineering and Testing Support at Securities Industry Automation Corporation where he manages the process improvement program and various testing activities for critical New York Stock Exchange trading systems. He has been leading software process improvement efforts in the military, telecommunications, and financial sectors for the last 10 years. As a certified Capability Maturity Model® (CMM®) Lead Assessor, he has performed a number of CMM-based assessments and provided process improvement guidance for various information technology organizations.

Securities Industry Automation Corporation
2 MetroTech CTR
Brooklyn, NY 11201
Phone: (212) 383-2963
Fax: (718) 923-6068
E-mail: sboycan@siac.com

Yuri Chernak, Ph.D., is the president and principal consultant of Valley Forge Consulting, Inc. As a consultant, Chernak has worked for a number of major financial firms in New York helping senior management improve their software testing process. Currently, his research interests focus on use-case-driven testing and test process assessment and improvement. Chernak is a member of the Institute of Electrical and Electronics Engineers (IEEE) Computer Society. He has been a speaker at several international conferences and has published papers on software testing in the IEEE publications and other professional journals. Chernak has a doctorate in computer science.

Valley Forge Consulting, Inc.
233 Cambridge Oaks ST
Park Ridge, NJ 07656
Phone: (201) 307-4802
Fax: (201) 307-4803
E-mail: ychernak@yahoo.com

CALL FOR ARTICLES

If your experience or research has produced information that could be useful to others, CROSSTALK can get the word out. We are specifically looking for articles on software-related topics to supplement upcoming theme issues. Below is the submittal schedule for three areas of emphasis we are looking for:

Total Creation of a Software Project
December 2005
Submission Deadline: July 18

Communications
January 2006
Submission Deadline: August 22

What Is Up and Coming?
February 2006
Submission Deadline: September 19

Please follow the Author Guidelines for CROSSTALK, available on the Internet at <www.stsc.hill.af.mil/crosstalk>. We accept article submissions on all software-related topics at any time, along with Letters to the Editor and BackTalk.

COMING EVENTS

August 15-17
International Conference on Information Reuse and Integration: Knowledge Acquisition and Management
Las Vegas, NV
www.cs.fiu.edu/IRI05

August 16-18
ICSEng '05 International Conference on Systems Engineering
Las Vegas, NV
www.icseng.info

August 22-26
Association for Computing Machinery SIGCOMM 2005
Philadelphia, PA
www.acm.org/sigs/sigcomm/sigcomm2005/index.html

September 12-16
Practical Software Quality and Testing Conference 2005 North
Minneapolis, MN
www.psqtconference.com/2005north

September 18-23
International Function Point Users Group 1st Annual International Software Measurement and Analysis Conference
New Orleans, LA
www.ifpug.org/conferences/annual.htm

September 19-22
Better Software Conference and Expo 2005
San Francisco, CA
www.sqe.com/bettersoftwareconf

September 26-27
PDF Conference and Expo
Washington, DC
www.pdfconference.com

May 1-4, 2006
2006 Systems and Software Technology Conference
Salt Lake City, UT
www.stc-online.org